Multi-threaded ptlrpcd

We want to implement general ptlrpcd pool. The ptlrpcd threads in such pool are shared by all asynchronous RPCs on the client/server, like asynchronous glimpse lock, statahead, BRW request, data checksum, and so on.

1. Background

In old Lustre releases, like Lustre-1.x and Lustre-2.0, there were only two portal RPC daemons on each client/server, one served all asynchronous I/O RPCs, the other was for all other asynchronous non-I/O RPCs. Such load model cannot well use CPUs capacity on large SMP node, then cause some load unbalance issues, especially for the client under heavy I/O mode or traversing large directory with AGL (asynchronous glimpse lock) introduced, the CPUs taken by portal RPC daemons are always quite busy, almost 100% loaded, but other CPUs maybe idle. Under such case, transferring some load to other CPUs is quite necessary.

2. Functional specifications

We want multiple CPU cores to share asynchronous RPC load. So we start many ptlrpcd threads. But it is not more threads better. Because there are many other threads on the client/server already, and especially for client, there will be some unpredictable application threads. Too much ptlrpcd threads cannot help to improve performance more, but maybe increase overhead of system scheduling. So we allow the administrator to specify how many ptlrpcd threads to be started according to the real cases when loads modules. The ptlrpc parameter (some kind of tunable interface) "**max_ptlrpcds**" is just for that. As following:

insmod ptlrpcd.ko max_ptlrpcds=xxx

If the parameter "**max_ptlrpcds**" is not specified, the default count of started ptlrpcd threads will be equal to the CPU cores (hyper-thread) on the node. Means one ptlrpcd thread per CPU core. On the other hand, the ptlrpcd thread count is greater or equal 2 always, even thought you specify "max_ptlrpcds" as smaller, to guarantee that the worst case of introducing multi-threaded ptlrpcd will not be worse than the original case.

2.1. ptlrpcd bind policy

With multiple ptlrpcd threads running on multiple CPU cores, we need to consider the overhead caused by transferring ptlrpcd data cross-CPU cores, especially for large NUMA node. We can bind the ptlrpcd thread to the CPU core. But binding all ptlrpcd threads maybe cause response delay, because some CPU core(s) may be busy with other things as to cannot process some asynchronous RPCs in time.

For example: "ls -l", some asynchronous RPCs for statahead are assigned to 'ptlrpcd_0', and 'ptlrpcd_0' is bound to 'CPU_0', but 'CPU_0' may be quite busy with other non-ptlrpcd things, like "ls -l" itself (we want to the "ls -l" thread, statahead thread, and ptlrpcd thread can run in parallel). Under such case, the statahead asynchronous RPCs cannot be processed in time, which is unexpected. If 'ptlrpcd_0' can be re-scheduled on other CPU core, it may be better.

So we shouldn't be blind for avoiding the data transfer. We make some compromise: divide the ptlrpcd threads pool into two parts. One part is for bound mode, each ptlrpcd thread in this part is bound to some CPU core. The other part is for free mode, and all the ptlrpcd threads in the part can be scheduled on any CPU core. We specify some partnership between bound mode ptlrpcd thread(s) and free mode ptlrpcd thread(s), and the asynchronous RPC load within the partners is shared.

It can partly avoid data transferring cross-CPU (if the bound mode ptlrpcd thread can be scheduled in time), and also can try to process asynchronous RPC ASAP (as long as the free mode ptlrpcd thread can be scheduled on any CPU core). We support the following ptlrpcd bind policy:

1) PDB_POLICY_NONE = 1
All ptlrpcd threads are free mode, and each ptlrpcd thread can be scheduled on any CPU core.

2) PDB_POLICY_FULL = 2
All ptlrpcd threads are bound mode. Try to bind every ptlrpcd thread to some CPU core evenly.

3) PDB_POLICY_PAIR = 3
Every two ptlrpcd threads compose one pair. In the pair, one ptlrpcd thread is free mode, the other one is bound mode, and they are partners for each other. Each free mode ptlrpcd thread has and only has one bound mode partner. Each bound mode ptlrpcd thread has and only has one free mode partner. The partner mode is something like: <free1, bound1> <free2, bound2> ... <freeN, boundN>

4) PDB_POLICY_NEIGHBOR = 4
Every two ptlrpcd threads compose one pair. In the pair, one ptlrpcd thread is free mode, the other one is bound mode, and they are partners for each other. Each free mode ptlrpcd thread has two bound mode partners. Each bound mode ptlrpcd thread has two free mode partners. The partner mode is something like: <free1, bound1> <bound1, free2> ... <freeN, boundN> <boundN, free1>

We allow the administrator to specify the ptlrpcd bind policy according to the real cases when loads modules. The ptlrpc parameter (some kind of tunable interface) "**ptlrpcd_bind_policy**" is just for that. As following:

insmod ptlrpcd.ko ptlrpcd_bind_policy=xxx

If the parameter “ptlrpcd_bind_policy” is not specified, “**PDB_POLICY_PAIR**” will be used as the default bind policy. It may be not the best one, but it is simple and suitable for most cases. In the future, we can support some more complex policy/partnership based on the patches for CPU partition.

2.2. ptlrpcd load policy

Although there are multiple ptlrpcd threads, for each asynchronous RPC, it only can be processed by one ptlrpcd. It is the RPC sponsor’s duty to specify how to push the asynchronous RPC into some ptlrpcd queue, but it is not enforced, and affected by "ptlrpcd_bind_policy". If the bind policy is "**PDB_POLICY_FULL**", then the RPC will be processed by the selected ptlrpcd. Otherwise, the RPC may be processed by either the selected ptlrpcd or one of its partner, depends on which is scheduled firstly, to accelerate the RPC processing. We support the following ptlrpcd load policy:

1) PDL_POLICY_SAME = 1
The sponsor prefers to the asynchronous RPC processed by the ptlrpcd thread that is running on the CPU core on which the sponsor is running, or the current CPU core.

2) PDL_POLICY_LOCAL = 2
The sponsor prefers to the asynchronous RPC processed by the ptlrpcd thread that is running on one of the CPU cores, such CPU core and the current CPU core belong to the same CPU partition, but it is different from the current CPU core.

3) PDL_POLICY_ROUND = 3
Use round-robin algorithm to dispatch the asynchronous RPCs to every ptlrpcd thread in turn for trying to balance the asynchronous RPC load among the whole system. It does not care on which CPU core the selected ptlrpcd is running, as long as it is not the current CPU core.

4) PDL_POLICY_PREFERRED = 4
The sponsor uses its own algorithm to specify the ptlrpcd thread to process the asynchronous RPC(s). But just as explained above, it not enforced if the ptlrpcd bind policy is not “**PDB_POLICY_FULL**”.

When the asynchronous RPC sponsor does not know which load policy to be used, it can consider to use “**PDL_POLICY_LOCAL**” or “**PDL_POLICY_ROUND**”, they are the usually used ptlrpcd load policies.